International Journal of Smart Computing and Artificial Intelligence
International Institute of Applied Informatics
2018, Vol.2, No.1, P.23 - 42

A Method to Adapt Storage Protocol Stack Using Custom File Metadata to Commodity Linux Servers

Takayuki Fukatani*,
Atsushi Sutoh*, Takahiro Nakano*

Abstract

Storage vendors have had difficulty adapting their proprietary protocol stacks to commodity servers. Despite large development effort, many proprietary protocol stacks can be used only on the specialized storage appliances because they depend on specialized hardware or software. Most of these protocol stacks use original filesystems to store custom file metadata that contain the protocol specific metadata to comply with the protocol specifications. In this study, we propose a new metadata management module named the protocol metadata module (PMM) that enables the custom file metadata to be used on commodity Linux servers. The PMM uses a portable operating system interface (POSIX)-based Linux interface to store the custom file metadata in Linux filesystems so that the protocol stacks can use the custom file metadata without the specialized hardware or software. The PMM enables the protocol stacks using the custom file metadata to offer the protocol functions on commodity Linux servers. We applied our developed PMM to the protocol stack of our storage appliance to verify its concept. Our evaluation results show that the PMM increases the protocol function coverage from 75.0% to 96.7% on Linux servers while suppressing the access performance degradation to at most 8% in the typical file server workloads.

Keywords: file server, POSIX, portability, Linux

1. Introduction

For many years, storage vendors have developed specialized storage appliances to satisfy user demands for performance, functionality, and reliability [1][2]. Conventionally, software-based protocol stacks process client requests such as a network file system (NFS), server message block (SMB), and Internet small computer interface (iSCSI) on those appliances [3][4]. The vendors have made large effort to incorporate frequent updates in the protocol specifications into their protocol stacks.

* Hitachi Ltd., Kanagawa, Japan

However, the conventional storage vendors have struggled to adapt their protocol stacks to the commodity servers. As interest in software-defined storage has increased [5][6], users have demanded cheaper commodity servers for storage hardware for light-weight workloads. To achieve this transition, some storage vendors use a hypervisor-based virtual machine [2]. However, this approach is not applicable to the protocol stacks that depend on specialized hardware.

Many storage appliances use their original filesystems to store user files and custom file metadata that contain protocol specific metadata [1][2]. These original filesystems are often implemented in specialized hardware or software. The custom file metadata enable the protocol stacks to offer protocol functions in a protocol compliant manner. However, the protocol stacks cannot be used on other filesystems because they depend on the original filesystems.

The use of the standardized portable operating system interface (POSIX) filesystem removes the dependency on the specialized hardware or software from the protocol stacks [7]. The POSIX filesystem is used in a wide range of open-source software (OSS) filesystems. Once the protocol stacks support the POSIX filesystem, they can be used on the commodity servers on which OSS filesystems are deployable.

However, on the POSIX-based filesystems, the protocol stacks cannot offer protocol functions that rely on the custom file metadata. The POSIX-based filesystems store the file metadata in the POSIX compliant format that does not cover protocol specific metadata. The protocol function coverage of the protocol stacks largely decreases on Linux servers because of the lack of the custom file metadata.

In this study, we propose a protocol metadata module (PMM) that enables the custom file metadata to be used on OSS Linux[i] filesystems [8]. It utilizes a POSIX-based Linux application programming interface (API) to store the custom file metadata in Linux filesystems so that the protocol stacks can use the custom file metadata on Linux servers.

The PMM consists of two components: the metadata management that stores the custom file metadata in Linux filesystems, and the journal management that ensures data integrity of the custom file metadata [9]. These modules complement the differences between the custom file metadata and the POSIX file metadata.

We applied our PMM to the protocol stack of our storage appliance named High-performance Network Attached Storage (HNAS). The PMM enables the HNAS protocol stack to offer protocol functions that rely on the custom file metadata on Linux servers.

Our evaluations show that the PMM increases the functional coverage of the HNAS protocol stack from 75.0% to 96.7% on Linux servers while suppressing the performance degradation to at most 8% in the typical file server workload. We found that the HNAS protocol stack with PMM achieves a file server performance competitive against other OSS protocol stack implementations. Also, we found that the journal management of the PMM ensures data integrity of the custom file metadata by offering fast and scalable recovery processing upon a system failure.

2. Background and Objective

2.1. HNAS Protocol Stack

Many proprietary protocol stacks use specialized hardware or software to store the custom file metadata to comply with the protocol specifications. This hardware or software usually has the original interface and functionality.

HNAS, which is the target storage appliance of this study, uses an original field-programmable gate array (FPGA)-based filesystem [10]. Like many other storage appliances, HNAS uses the original filesystem to store the custom file metadata in a protocol compliant format. The software-based protocol stack processes client requests that involve advanced file processing such as the security descriptor, the named stream, Quota, and the virus scan [11]. The HNAS protocol stack uses the custom file metadata to provide these capabilities. Figure 1 shows an over-view of the HNAS architecture.

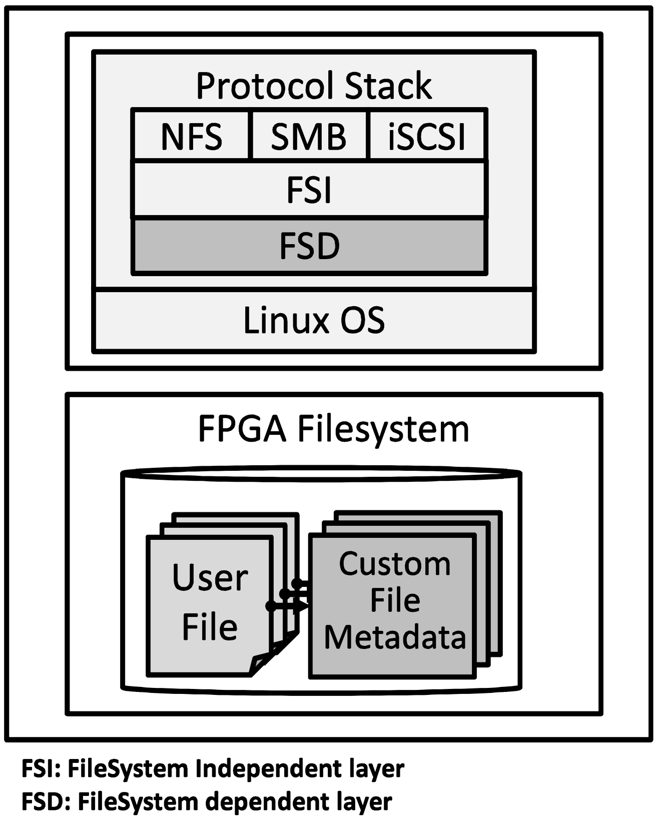

Figure 1 HNAS Overview

The HNAS protocol stack has a layered architecture that separates hardware dependent modules from other modules. The architecture consists of the protocol layer, the FileSystem Independent (FSI) layer, and the FileSystem Dependent (FSD) layer. The protocol layer accommodates NFS, SMB, and iSCSI modules, which interpret client requests. Underneath the protocol layer, the FSI layer offers the aforemen-tioned advanced filesystem functions. The FSD layer encapsulates the FPGA filesystem implementation from the upper layers.

Although a large part of the HNAS protocol stack is independent of the FPGA filesystem, the protocol stack runs only on the HNAS appliance because it depends on the FPGA filesystem in the FSD layer.

2.2. Objective

Despite the large development effort, the proprietary protocol stacks can be used only on specialized appliances and not on other platforms because they depend on the original filesystems. If the protocol stacks are enabled to run using alternative filesystems, users will be able to choose a wider range of configurations on the basis of workload requirements.

Use of the POSIX filesystem removes the dependency on the specialized appliance from the protocol stacks. The POSIX filesystem is widely used in OSS filesystems such as Ext4/XFS [12][13]. Once a protocol stack supports the POSIX filesystem, it can be used on commodity Linux servers on which the OSS filesystem is commonly used.

However, to comply with the protocol specifications, the protocol stacks need the custom file metadata originally stored in the original filesystems. Without the custom file metadata, the protocol stacks cannot offer the protocol functions that require the protocol specific file metadata.

This study aims to adapt the protocol stacks using the custom file metadata to commodity Linux servers. Our developed PMM enables the custom file metadata to be used on Linux servers. We use the POSIX-based Linux API in the PMM so that the protocol stacks can use the custom file metadata on Linux servers.

Our PMM aims to enable the protocol stacks to offer the protocol functions that rely on the custom file metadata on Linux servers. Along with the higher protocol func-tion coverage, the PMM aims to offer reasonable access performance and data in-tegrity assurance. Our performance target is the HNAS protocol stack that performs competitively against other OSS protocol stack implementations for a typical file server workload. Also, the PMM aims to enable fast recovery processing of data integrity of the custom file metadata upon a system failure.

3. Protocol Metadata Module

3.1. Challenges

Table 1 Linux File Metadata and Custom File Metadata

Items		Linux File Metadata	Custom File Metadata (HNAS)
Metadata Types	File Attributes	POSIX / Extended Attributes	SMB / NFS
	Security	POSIX Permission / ACL	SMB / NFS
	Named Stream	None	SMB / NFSv4
	Quota	Filesystem Function	Protocol Stack Function
Data Integrity Assurance		Per System Call	Per NFS / SMB Request

The PMM has to complement the differences between the POSIX-based Linux file metadata and the custom file metadata to comply with the protocol specifications. The PMM stores the user data and the file metadata in Linux filesystems. However, Linux filesystems do not support the protocol specific file metadata whereas the custom file metadata contain the metadata.

The differences between the Linux file metadata and the custom file metadata appear in metadata types and data integrity assurance. The custom file metadata comply with the NFS and SMB protocol specifications whereas the Linux file metadata comply with the POSIX standard. Although a large part of the NFS requirements overlaps the POSIX standard, SMB requires different types of file metadata from the POSIX standard because of the different operating system (OS) environment. Also, the protocol specifications require data integrity assurance on a per client request basis whereas POSIX ensures data integrity on a per system call basis [8].

Table 1 shows the differences between the Linux file metadata and the custom file metadata of HNAS.

For the metadata types, the Linux file metadata use necessary file attributes and security metadata for UNIX applications. Also, modern Linux file systems provide the extended attributes to add small file attributes to user files [12][13], and most Linux filesystems use original Quota databases to manage filesystem usage.

On the other hand, the HNAS custom file metadata comply with the SMB and NFS specifications [3][4]. The HNAS custom file metadata use an additional set of file attributes such as the file creation time or the archive attribute. The security metadata contain the security descriptors of multiple owner files [11]. The named stream stores arbitrarily sized named user data. The Quota database stores the Quota entries given by the HNAS protocol stack.

For the data integrity assurance, modern Linux filesystems use the journaling systems to ensure the data integrity. These journaling systems guarantee the data integrity on a per system call basis, so they do not ensure the data integrity among multiple system calls [14]. Therefore, if a system failure happens between a user file update and an update of the custom file metadata stored in the extended attributes or system files, the data integrity between the user file and the custom file metadata is not guaranteed. In contrast to Linux metadata, HNAS guarantees the atomicity of all updates during processing of a single client request. The above differences cause less protocol function coverage and a lack of data integrity assurance when the protocol stacks using the custom file metadata run on Linux filesystems.

The PMM needs to complement these differences to achieve the objectives. To offer higher protocol function coverage, the PMM needs the metadata of the protocol functions such as the SMB and NFS file attributes, the security descriptor, the named stream, and Quota. The PMM also needs to assure the atomicity of updates of user data and metadata during a single client request. And also, the performance is also a concern if the PMM introduces new metadata or data integrity assurance method. These modifications are usually accompanied by performance overhead such as additional metadata accesses and the journal logging [18][19][25][26].

3.2. PMM Architecture

Our PMM introduces a new metadata management and the journal management to realize the custom file metadata and the data integrity assurance on Linux filesystems. The metadata management stores the custom file metadata in Linux file systems to enable the custom file metadata to be used on Linux servers. The journal management records the journal logs in the journal file to ensure the data integrity between user files and the custom file metadata. In addition, performance optimization techniques are used in the new modules to achieve the reasonable performance.

We implemented our PMM in the HNAS protocol stack. The PMM works as an internal module in the new FSD module named Linux FSD. Linux FSD supports XFS and Ext4. An interface called an FSD API clearly separates the FSD layer from the FSI layer in the HNAS protocol stack. The compliance with the FSD API in Linux FSD enables the protocol layer and the FSI layer to work on Linux servers. Figure 2 shows the PMM overview.

Linux FSD maps an FSD API call to system calls for file operations. The file operations in the FSD API are mostly compatible with Linux system calls including open, read, write, and so on [8]. Therefore, the API mapping between the FSD API and system calls for the file operations is straightforward.

Along with the file operations, the PMM issues system calls for the metadata manage-ment and the journal management. In contrast to the file operations, the metadata man-agement and the journal management are the original functions of the PMM. These modules issue the system calls in the performance optimized way. We describe the de-tails in the next section.

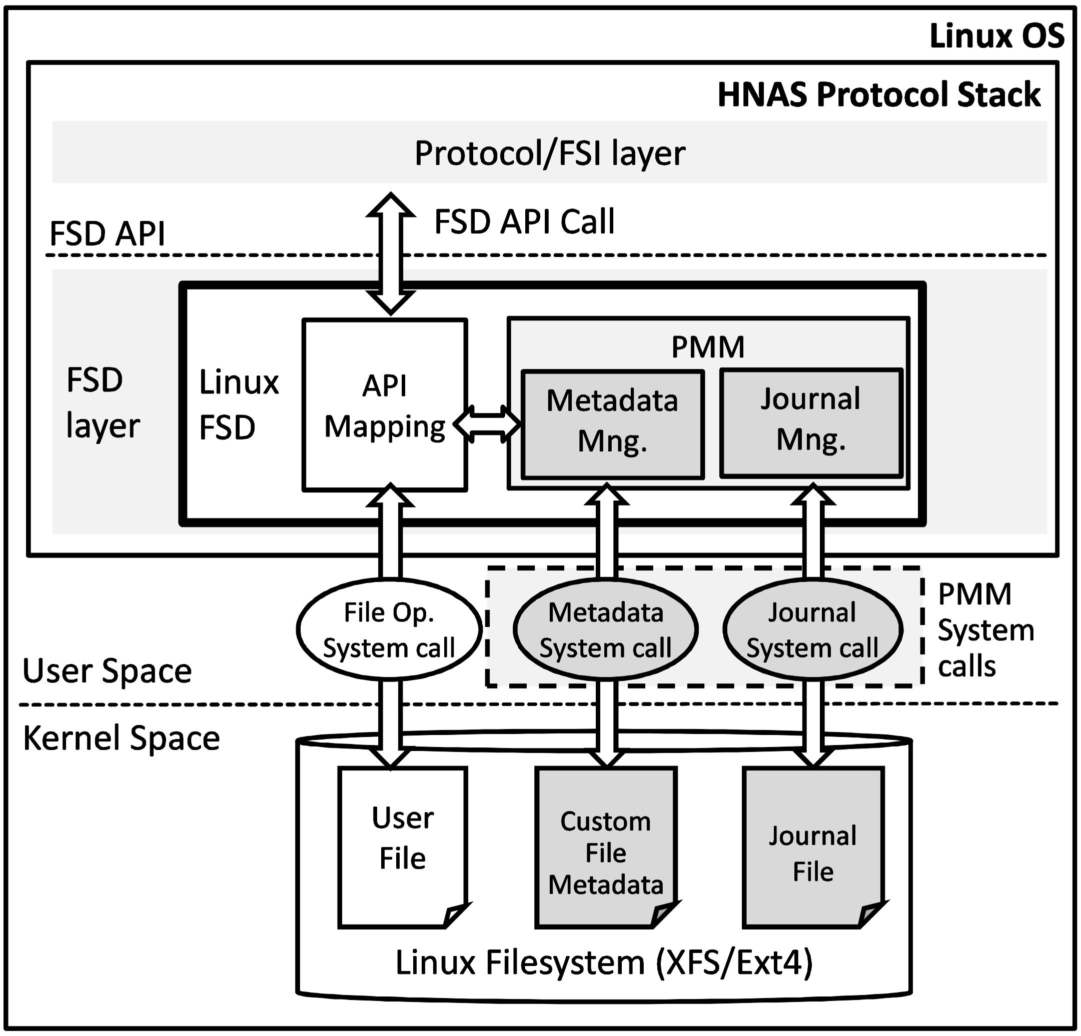

Figure 2 PMM Overview

4. Implementations

4.1. Metadata Management

The implementation of the metadata management consists of the namespace management, the extended attributes, and the open_by_handle system call [15]. The PMM divides the namespace of Linux filesystems into the system directory and the data directory. The PMM uses the extended attributes to store file attributes with a fixed size and file handles of metadata files that contain arbitrarily sized metadata. The PMM also uses the file handles and the open_by_handle system call to link the metadata files to one or more user files. These data structures enable the custom file metadata with an arbitrary size or multiple-owner files to be used in Linux filesystems. Figure 3 shows the metadata layout in Linux filesystems.

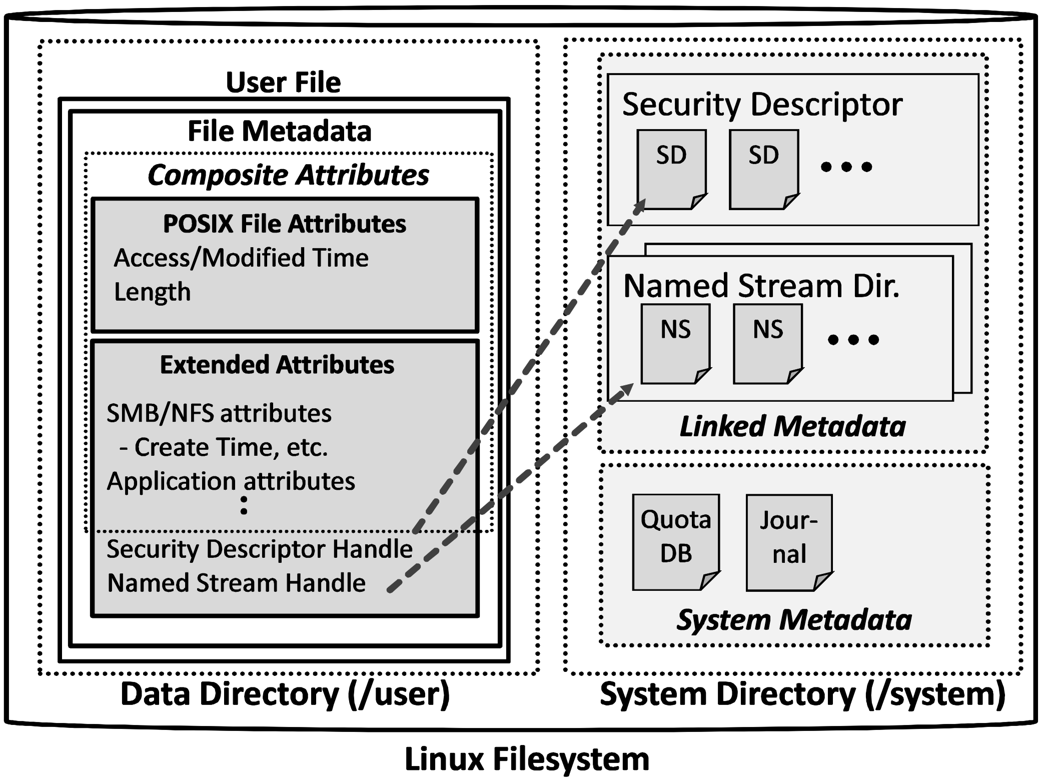

Figure 3: Metadata layout in Linux filesystems

The PMM offers three types of file metadata: composite attributes, linked metadata, and system metadata. We explain these file metadata below.

4.1.1. *Composite attributes*

The PMM uses the composite attributes to offer predefined file attributes. The composite attributes contain SMB/NFS compatible file attributes and application attributes used in the HNAS protocol stack.

The storage of the composite attributes consists of the extended attributes and the POSIX file attributes of user files. The PMM uses the extended attributes to store most of the file attributes of the custom file metadata. Additionally, the PMM uses a small part of the POSIX file attributes and converts them into a compatible format with the custom file metadata. These POSIX file attributes include the access time, the modified time, and the file length, which are updated during write system calls. The use of the POSIX file attributes eliminates the disk access to the extended attributes in the write request processing and improves write performance. We evaluate the performance improvement in the next section.

4.1.2. *Linked metadata*

The linked metadata store the custom file metadata that have arbitrary sizes and multiple-owner files. The PMM uses the linked metadata to store the named stream without the size limitation. Also, the capability of the multiple-owner files reduces the storage capacity of duplicated security descriptors like Windows NTFS [11].

The PMM creates metadata files or directories in the system directory for all linked metadata. The PMM then records the file handles of the linked metadata files and directories to the extended attributes of owner files. It uses the file handles and the open_by_handle system call to access the linked metadata. The PMM manages the reference count of the linked metadata in the extended attributes of the linked metadata file. It then deletes the linked metadata file when the reference count becomes zero to prevent a linked metadata file from becoming an orphan. In addition, the PMM creates a metadata directory when the first named stream of a user file is created. The PMM stores named streams of the user file to the same directory.

4.1.3. System metadata

The system metadata contain the filesystem metadata such as the Quota database and the journal log, which is described in the next subsection. The system metadata are system files with pre-defined pathnames in the system directory. The PMM provides an access interface for the system metadata files to the FSI layer through the FSD API. The FSI layer stores arbitrary contents to the system metadata.

4.2. Journal Management

The journal management of the PMM ensures data integrity between user files and the custom file metadata. The PMM introduces user-space journaling to extend the scope of data integrity assurance from a per system-call basis to a per client request basis.

The custom file metadata could be inconsistent with user files without the journal management. The PMM issues system calls to update user files and the custom file metadata separately. Without any additional protections, the user files and the custom file metadata could be inconsistent when a system failure happens between these system calls. These inconsistencies include inconsistent file attributes, inconsistent security descriptors or named stream handles, and inconsistent linked metadata reference count. These inconsistencies could cause invalid file metadata, security violation, orphan metadata files, or unexpected system behaviors.

To solve the above inconsistencies, the PMM offers journaling and replay processes. We describe both processes below.

4.2.1. Journaling

The PMM manages a system file named the journal file to record all metadata updates as journal logs. The PMM uses several types of journal logs corresponding to the above inconsistencies. A journal log consists of the log header and the updated metadata. A log header contains the necessary information for the replay process such as the log type, the sequential number, the update time, the file handle of a target user file, and request type. Figure 4 shows an example journal log.

<<Log header>>

Log Type: Composite Attributes
Sequential Number: 0x001
Update Time: 2016-01-29 12:00:01
Request Type: Write
File handle: 0x00001

<<Log body>>

(Updated contents of the custom file metadata)

Figure 4: Example of Journal Log

For each client request, the PMM flushes journal logs to the journal file before issuing any system calls that update the target user files. An internal service thread, which processes client requests, adds the journal logs to a per-thread log buffer on memory. The PMM merges the journal logs into a single disk write and flushes them to the journal file. The PMM periodically restarts the journal file every few seconds so that the journal file does not consume large disk capacity. Figure 5 shows the overview of the journaling processing.

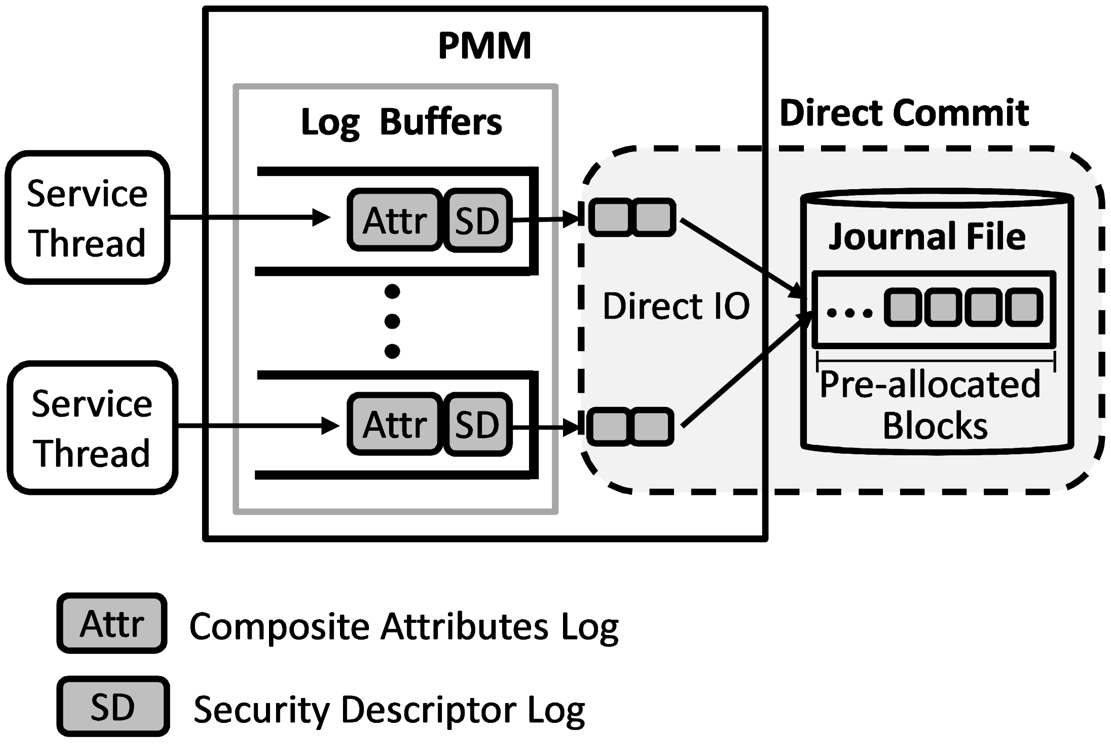

Figure 5: Journaling in PMM

The PMM uses an optimization named the direct commit to reduce the journaling overhead. User-space journaling is known to cause a large delay due to the kernel journaling during synchronous journal file updates [16]. To avoid the delay, the PMM pre-allocates the disk blocks of the journal file. Also, the PMM uses Direct IO to update the journal file to reduce the page cache overhead in the kernel space [8]. The PMM uses a predefined journal file size that is calculated from the journal log size and the required system throughput. If the total size of journal logs on the journal file exceeds the predefined file size, the PMM appends the extra log items to the end of the journal file and truncates the journal file to the predefined size when the PMM restarts the journal file.

4.2.2. Replay

When a system failure happens, the PMM starts the replay process in the next system reboot. In the replay process, the PMM reads the journal logs on the journal file while the log headers contain the continuous sequential numbers.

For each journal log, the PMM checks the existence and the modified time of the target user file given in the file handle field of the log header. The PMM uses the open_by_handle system call to find the target user file. If the target user file is created, deleted, or modified after the update time in the log header, the PMM reflects the updated metadata to the target metadata on the basis of the log type. Otherwise, the PMM discards the journal log because the system calls that update the target user files do not take effect during the system failure.

5. Evaluations

5.1. Protocol Function Coverage

We evaluated the protocol function coverage of the HNAS protocol stack on Linux servers. We listed 60 HNAS protocol functions from SMB, NFS, and iSCSI specifications [3][4]. The SMB includes not only SMB specifications but also other specifications used in the SMB environment such as Volume Shadow Copy Service (VSS). Also, NFS and iSCSI include other related specifications

We evaluated the protocol function coverage without and with the PMM (wo PMM and w PMM). Table 2 shows the evaluation results.

The PMM improves the functional coverage from 75.0% to 96.7%. The security descriptor enables the SMB security descriptor, NFSv4 ACL, and mixed mode security. The named stream enables the SMB named stream, SMB2 symbolic link, SMB3 transparent failover, and NFSv4 named attribute. The Quota database enables the NFSv3 rquota and NFSv4 quota. The composite attributes enable other improvements.

The PMM does not cover the SMB long file name or volume shadow copy because of the lack of corresponding functions in Linux filesystems. The PMM is supposed to be capable of these features if Linux filesystems or other modules provide the correspond-ing functions.

Table 2: Protocol Function Coverage Without and With PMM

Spec.	Function name	woPMM	wPMM	Spec.	Function name	woPMM	wPMM	Spec.	Function name	woPMM	wPMM
SMB1	SMB command set	Yes	Yes	SMB2	Message signing(SHA-256)	Yes	Yes	NFS3	NLM/NSM	Yes	Yes
	User authentication	Yes	Yes		Asynchronous operatio	Yes	Yes		portmap	Yes	Yes
	Remote administration	Yes	Yes		Credit for flow control	Yes	Yes		mountd	Yes	Yes
	Remote procedure call	Yes	Yes	SMB 2.1	File lease	Yes	Yes		rquota	No	Yes
	Named pipe	Yes	Yes		Resilient file handles	Yes	Yes	NFS4	NFSv4 command set	Yes	Yes
	Security descriptor	No	Yes		Reauthenticate	Yes	Yes		Mandatory auth.	Yes	Yes
	Oplock	Yes	Yes		Dialect negotiation	Yes	Yes		UCS file name	Yes	Yes
	File change notificatio	Yes	Yes	SMB3	Transparent failover	No	Yes		NFSv4 ACLs	No	Yes
	Distributed file system	Yes	Yes		Multi-channel	Yes	Yes		File delegation	Yes	Yes
	SMB file attributes	No	Yes		Witness	Yes	Yes		Compounded request	Yes	Yes
	Unicode file name	Yes	Yes		Signing(AES-CMAC)	Yes	Yes		Named attribute	No	Yes
	Long file name	No	No	SMB Other	Remote shared virtual disk	Yes	Yes		quota	No	Yes
	8.3 name	No	Yes		Volume shadow copy	No	No		Pseudo-fs	Yes	Yes
	Case insensitive FN	No	Yes		Access-based enum.	Yes	Yes	NFS/ SMB	Mixed mode security	No	Yes
	Named stream	No	Yes		Auditing event log	Yes	Yes		Inter-protocol file locl	Yes	Yes
	Singning(MD5)	Yes	Yes		Virus scan	No	Yes	iSCSI	SCSI access	Yes	Yes
SMB2	SMB2 command set	Yes	Yes	NFS3	NFSv3 command set	Yes	Yes		Persistent reservation	Yes	Yes
	Durable open	Yes	Yes		Unix permissions	Yes	Yes		Multi-path	Yes	Yes
	Symbolic link	No	Yes		DES encrypted auth.	Yes	Yes		Path load balancing	Yes	Yes
	Compounded request	Yes	Yes		Kerberos	Yes	Yes		iSNS	Yes	Yes
Functional coverage: Without PMM 45/60 (75.0%), With Protocol Metadata 58/60 (96.7%)											

5.2. Access Performance

5.2.1. Measurement Environment

We used a commodity server (Dell[ii] R610) as an NFS server in the following evaluations. We used another server with the same configuration as an NFSv3 client. The client accesses the server via a 10 Gbps network. We used a serial ATA (SATA) disk on the server so that we can evaluate the impact of the metadata access increase of the PMM in the disk bottleneck environment. Table 3 shows the evaluation configuration.

Table 3 Evaluation Configuration

Item		Description
Hardware (Server and Client)	CPU	Intel Xeon E5-2620 [iii](6 cores, 2.0 GHz)
	RAM	12 GB
	Storage	Seagate Constellation.2 ST9500620NS SATA 6 Gb/s, 500 GB 7200 rpm
	NIC	Broadcom NetXtreme II, BCM57810 10 Gigabit Ethernet
Software (Server)	OS	Debian Linux Wheezy (kernel 3.16.0)(kernel 3.16.0)
	Filesystem	XFS (inode size 512 bytes)
	NFS server	HNAS protocol stack with PMM, kernel nfs daemon (knfsd), NFS ganesha v4.1
Software (Client)	OS	Debian Linux Wheezy (kernel 3.16.0)
	Benchmark	filebench v 1.4.9.1

5.2.2. Measurement Results

We used filebench [17] to measure the performance of the typical file server performances including read, write, file server, and mail server performance. We used the fivestreamwrite and fivestreamread workloads of filebench to measure the read and write performance. We used the fileserver workload for the file server performance and the varmail workload for the mail server performance. The fileserver workload consists of create, write, open, append, read, close, delete, and stat to middle-sized files. The varmail workload consists of create, write, fsync, read, open and close to small files. In the measurements, the total file-set size is set at least two times larger than the server and client memory to avoid cache effects.

First, we evaluated the performance overhead of our PMM. We evaluated the performance of the HNAS protocol stack without the PMM (without PMM) and with it. In the evaluations with the PMM, we measured the performance without any optimizations (with PMM (No option)), with composite attribute optimization (with PMM (CA)), and with direct commit (with PMM (CA + DC)). Figure 6 shows the measurement results.

The optimizations suppress the performance degradation of PMM from 32% to 0% in the write performance, 16% to 7% in the file server performance, and 20% to 8% in the mail server performance. For the read performance, no overhead was found because the workload does not involve PMM overhead.

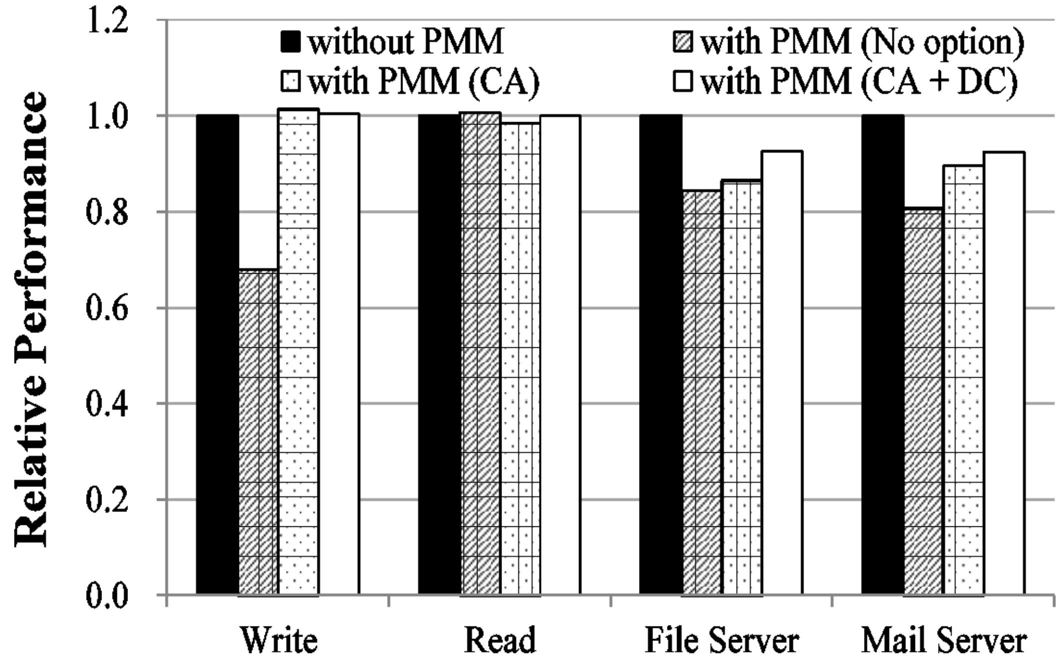

Figure 6: NFS Performance with and without optimizations

Table 4 Number of Disk Writes per Filebench Operation

	Write	*File server*	*Mail server*
without PMM	8.51	0.82	0.91
with PMM (No opt.)	25.14	1.29	1.46
with PMM (CA)	8.16	1.05	1.13
with PMM (CA + DC)	8.30	1.02	1.08

As shown in Table 4, the decrease in the number of disk writes for a single filebench operation explains why these performances improved. In the write workload, the composite attribute optimization reduces the number of disk writes by over 75%. In the write workload, filebench issues 1 MB write system calls, which are split into 64 KB NFS write requests by the NFS client module. If the composite attribute optimization is not applied, the PMM issues a disk write for updating the extended attributes in each 64 KB NFS write request. The composite attribute optimization eliminates these disk accesses during the NFS write request processing. In addition, 64 KB NFS write requests are sequentially processed and merged into a single 128 KB disk write by Linux OS [8]. Theoretically, the composite attribute optimization decreases the number of disk writes from 24 to 8 for a single 1 MB write operation.

Also, the direct commit reduces the number of disk writes in the file server workload and the mail server workload by 2% and 5%, respectively, by eliminating the overhead in the kernel journaling and the page cache management.

Second, we evaluated the performance of the HNAS protocol stack (Proposal) against NFS-ganesha [18] (ganesha) and the kernel nfs daemon (knfsd). As we can see in Figure 7, the HNAS protocol stack has 10% and 20% better write and read performances than OSSs, respectively.

The HNAS protocol stack issues disk IOs to a single file as continuously as possible to make the disk IOs more sequential. This optimization is supposed to contribute to better read and write performances than OSSs.

Even with the overhead of the PMM, the HNAS protocol stack performs equivalently to OSSs in the file server evaluation. As the file server workload uses 128 KB files on average, the better write and read performances are thought to contribute to this result.

On the other hand, the HNAS protocol stack shows 20% lower performance in the mail server evaluation that uses 16 KB file size. This result suggests that the HNAS protocol stack performs worse than OSSs for the metadata intensive workload because of the PMM overhead and its original performance characteristics.

As write, read, and file server workloads are the typical file server workloads, it can be said that PMM performs competitively against OSSs in the HNAS protocol stack.

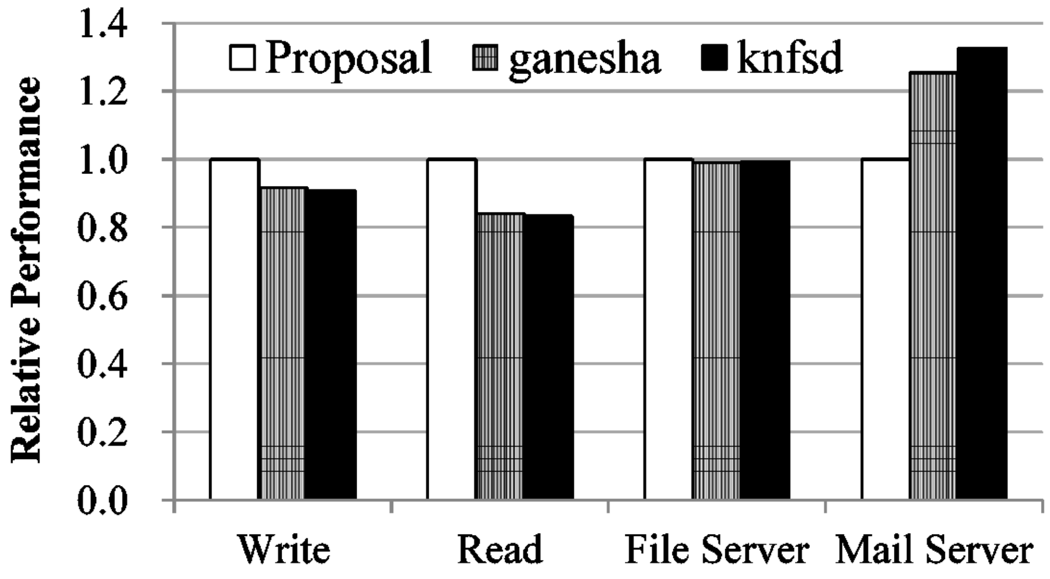

Figure 7: Performance comparison with OSS Implementations

5.3. Recovery Performance

We evaluated the recovery performance of the PMM journal management upon a system failure. In case of a system failure, the PMM recovers the data integrity of the custom metadata when the HNAS protocol stack mounts the failed filesystem. We evaluated the amount of time to mount the failed filesystem with and without the PMM journal management. Also, we evaluated the performance of the exhaustive check used in conventional filesystem utilities like fsck [9]. The exhaustive check scans all files in the failed filesystem and recovers the consistency of metadata if necessary.

In the evaluations, we intentionally caused system crashes while issuing file creation and deletion requests to the HNAS protocol stack. The evaluations were carried out with different numbers of stored files to see the scalability of the recovery methods.

Figure 8 shows the measurement results.

The mounting time with the PMM journal management was similar to that without it. The mounting time is only about two seconds when the filesystem contains one million files. The recovery process of PMM examines a small number of journal logs regardless of the total number of files in the filesystem. Therefore, the recovery time remains constant and short even if a filesystem contains a large number of files.

On the other hand, the mounting time of the exhaustive check increases in accordance with the number of files. The mounting time reaches nearly 100 seconds at one million files.

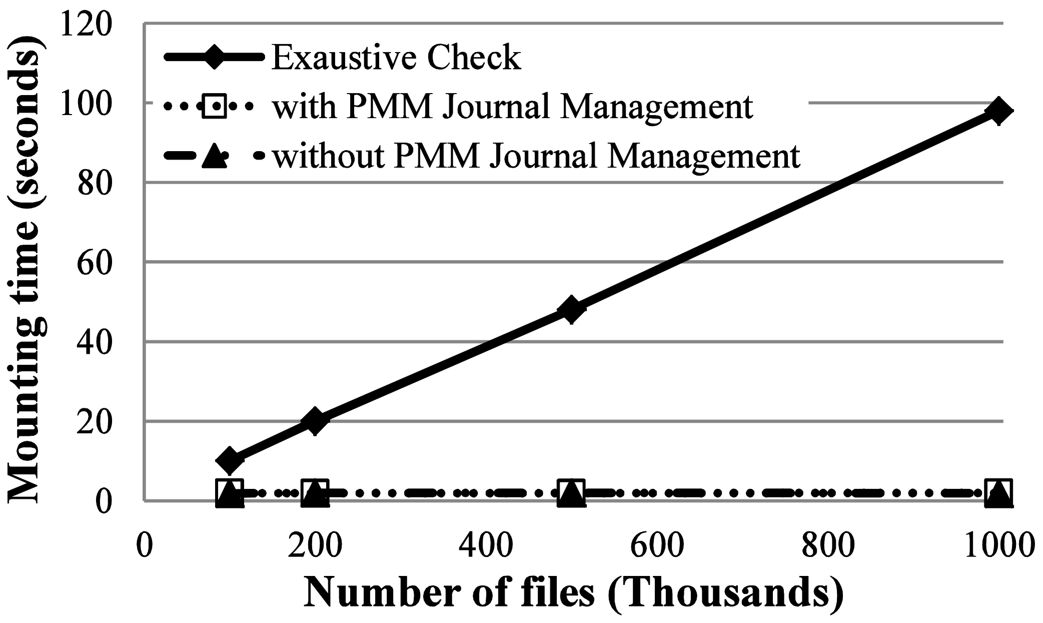

Figure 8: PMM Recovery Performance

These results show that the PMM journal management offers fast and scalable recovery upon a system failure.

6. Related Work

There have been a lot of studies to achieve higher protocol function coverage with less performance overhead on commodity servers. Basically, these studies are categorized into two approaches; the POSIX based approach or the VM based approach.

For the POSIX based approach, the OSS community has developed protocol stack implementations such as the kernel nfs daemon, NFS-ganesha, and samba [18][19]. Some of these implementations use the extended attributes and system files to store the protocol compliant file attributes or the named stream in a similar way to our proposed PMM. However, these OSSs do not ensure the data integrity of these custom file metadata upon system failures whereas the PMM does. Richacl stores the security descriptor to the extended attributes of each user file in Linux filesystems like the PMM does [20]. However, this approach imposes larger storage consumption of security descriptors than our multiple-owner metadata approach. Also, the PMM allows the reuse of the proprietary protocol stacks with the multiple protocol support and third-party vendor application support. These capabilities enable advanced protocol functions such as mixed mode security, the inter-protocol file lock, and the certified proprietary virus scan support whereas OSSs do not support these functions.

For the VM based approach, much research has focused on porting software of purpose-built appliances to commodity servers. Burtsev et al. proposed an efficient inter-VM communication for software of purpose-built NAS appliances running on virtual machines (VMs) [21]. Also, NetApp[iv] released the VM implementation of its NAS appliance to deploy the virtual appliance to the software-defined storage [22]. These studies focus on the protocol stacks using a software-based filesystem in the VM environment, so their methods are not applicable to the protocol stack using the hardware-based filesystem like HNAS.

The data integrity assurance of filesystems and its performance optimizations have long been studied [23][24]. For data integrity assurance in the kernel space, Verma et al. showed that kernel modifications enable user-space applications to ensure data integrity of application data [25]. However, these kernel changes can be done only in their filesystems. The PMM is applicable for Linux filesystems that are widely used. Moreover, user-space data integrity assurance has been widely studied in database research [26][27][28]. This study focuses on the application of the user-space data integrity assurance to the network storage protocol stack.

7. Conclusion

In this study, we proposed a method to adapt storage protocol stacks using the custom file metadata to commodity Linux servers. We developed a new metadata management module named the protocol metadata module (PMM), which enables the protocol stack to use the custom file metadata on Linux servers. We implemented the PMM in the protocol stack of our storage appliance, High-performance Network Attached Storage (HNAS), to prove the concept of the PMM.

The PMM utilizes Linux application programming interfaces (APIs) such as the namespace management, extended attributes, and open_by_handle system call to store the custom file metadata in Linux filesystems. The PMM enables protocol stacks to achieve higher protocol function coverage by using the custom file metadata on Linux servers

The PMM consists of metadata management and journal management. These new modules complement the differences between the custom file metadata and portable operating system interface (POSIX) file metadata. Also, we introduced performance optimizations to reduce the performance overhead of the PMM.

Our evaluations show that the PMM improves the coverage of the HNAS protocol functions from 75.0% to 96.2% on Linux servers. Our performance optimizations suppress the performance degradation of the PMM to up to 8% in the typical file server workloads. As a result, the HNAS protocol stack with the PMM performs competitively against OSS protocol stack implementations. Also, we found that the PMM journal management enables fast and scalable recovery processing for the data integrity assurance of the custom file metadata.

The PMM enables the proprietary protocol stacks to achieve high protocol function coverage while offering reasonable access performance and ensuring data integrity on commodity Linux servers.

References

[1] Watson, P. Benn, “Multiprotocol data access: NFS CIFS and HTTP,” Technical Report TR3014 Network Appliance, 1999.

[2] DELL EMC, “Dell EMC Isilon OneFS: A Technical Overview,” DELL EMC White paper, 2016.

[3] The Internet Engineering Task Force; http://www.ietf.org/

[4] Microsoft Developer Network; https://msdn.microsoft.com/

[5] M. Carlson, A. Yoder, L Schoeb, D. Deel, C.Pratt, “Software-defined storage,” Storage Networking Industry Association (SNIA), White paper, 2015.

[6] H. V. Madhyastha, J. C. McCullough, G. Porter, R. Kapoor, S. Savage, et. al. “scc: cluster storage provisioning informed by application characteristics and SLAs.” Proc. 10th USENIX Conference on File and Storage Technologies (FAST 2012), 2012.

[7] The Open Group. The Open Group Base Specifications Issue 7, 2013 Edition. http://pubs.opengroup.org/onlinepubs/9699919799/.

[8] D. P. Bovet, and M Cesati, “Understanding the Linux kernel,” 3rd Edition. O'Reilly Media, Inc., 2005.

[9] G. Sivathanu, CP. Wright, and E. Zadok, “Ensuring data integrity in storage: Techniques and applications,” Proc 2005 ACM workshop on Storage security and survivability, ACM, 2005.

[10] G. S. Barrall, T. Willis, S. Benham, M. Cooper, C. J. Aston, et al. “Apparatus and method for hardware implementation or acceleration of operating system functions,” U.S. Patent No 6,826,615, 2004.

[11] M. Russinovich, D, A. Solomon, and A. Ionescu, “Windows Internals (6th Edition),” Microsoft Press, 2012.

[12] xfs.org; http://xfs.org/.

[13] Ext4 Wiki; https://ext4.wiki.kernel.org/index.php/Main_Page.

[14] T. S. Pillai, V. Chidambaram, R. Alagappan, S. Al-Kiswany, A.C. Arpaci-Dusseau, et al., “All file systems are not created equal: on the complexity of crafting crash-consistent applications,” Proc. 11th Symposium on Operating Systems Design and Implementation (OSDI'14), 2014.

[15] open_by_handle(3) - Linux man page; https://linux.die.net/man/3/open_by_handle.

[16] W. Lee, K. Lee, and H Son. “Waldio: Eliminating the filesystem journaling in resolving the journaling of journal anomaly,” Proc. USENIX Annual Technical

Conference 2015 (ATC 2015), pp 235-247, 2015.

[17] V. Tarasov, E. Zadok, and S. Shepler. “Filebench: A Flexible Framework for File System Benchmarking,” article in ;login; magazine, Spring 2016, Vol. 41, No. 1

[18] P. Deniel, T. Leibovici, and J. C. Lafoucrière, “GANESHA, a multi-usage with large cache NFSv4 server,” Linux Symposium, 2007.

[19] samba.org; https://www.samba.org/.

[20] Kumar K. V, A. Grünbacher, and G. Banks. "Implementing an advanced access control model on Linux," Linux Symposium. 2010.

[21] A. Burtsev, K. Srinivasan, P. Radhakrishnan, L. N. Bairavasundaram, K. Voruganti, et al., “Fido: Fast Inter-Virtual-Machine Communication for Enterprise Appliances,” Proc. 2009 USENIX Annual Technical Conference (ATC 2009), 2009

[22] T. Pascu, “ONTAP Select Product Architecture and Best Practices,” NetApp Technical Report 4517, NetApp. 2016.

[23] J. Bornholt, A. Kaufmann, J. Li, A. Krishnamurthy, E. Torlak et al., “Specifying and Checking File System Crash-Consistency Models,” Proc. 21st International Conference on Architectural Support for Programming Languages and Operating Systems 2016 (ASPLOS '16), 2016.

[24] T. S. Pillai, V. Chidambaram, R. Alagappan, S. Al-Kiswany, A. C. Arpaci-Dusseau et al., “All file systems are not created equal: on the complexity of crafting crash-consistent applications,” Proc. 11th USENIX conference on Operating Systems Design and Implementation (OSDI'14), 2014.

[25] R. Verma, A. A. Mendez, S. Park, S. Mannarswamy, T. Kelly, et al., “Failure-atomic updates of application data in a linux file-system,” Proc. 13th USENIX Conference on File and Storage Technologies (FAST'15) 2015, pp 203-211, 2015.

[26] Atomic Commit In SQLite; https://www.sqlite.org/atomiccommit.html.

[27] M. Zheng, J. Tucek, D. Huang, F. Qin, M. Lillibridge et al., “Torturing databases for fun and profit,” Proc. 11th Symposium on Operating Systems Design and Implementation (OSDI'14), pages 449–464, 2014.

[28] T. Q. Dam, S. Cheon and Y. Won. “On the IO characteristics of the SQLite Transactions.” MOBILESoft 2016, 2016.

[i] Linux is a registered trademark of Linus Torvalds in the United States and other countries.
[ii] DELL is a trademark or a registered trademark of Dell EMC Corporation or its subsidiaries in the United States and other countries.
[iii] Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

[iv] NetApp is a trademark a trademark of NetApp, Inc.